

PC-8001B

Micro Computer

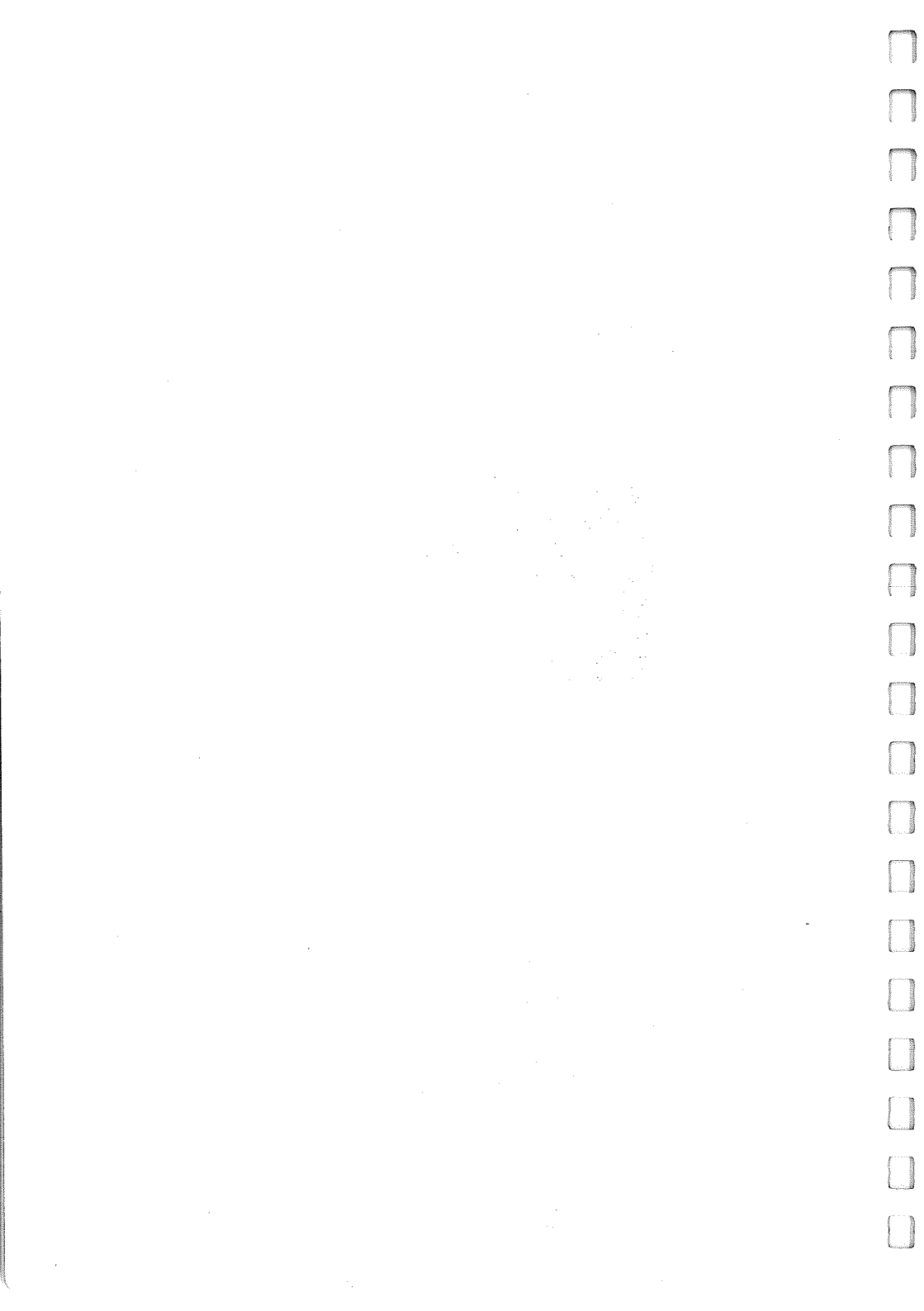
User's Manual



NEC

PC-8101B

PTS-068



PC-8001B

Micro Computer

User's Manual

NEC

IMPORTANT NOTICE

- (1) All rights reserved. This manual is protected by copyright. No part of this manual may be reproduced in any form whatsoever without the written permission of the copyright owner.
- (2) The policy of NEC being that of continuous product improvement, the contents of this manual are subject to change, from time to time, without notice.
- (3) All efforts have been made to ensure that the contents of this manual are correct; however, should any errors be detected, NEC would greatly appreciate being informed.
- (4) NEC can assume no responsibility for errors in this manual or their consequences.

© Copyright 1981 by Nippon Electric Co., Ltd.

TABLE OF CONTENTS

	Page
SECTION I — IN THE BEGINNING	1
Chapter 1 The PC-8001 System	3
1.1 PC-8000 System Components	4
1.2 PC-8000 System Peripherals	5
Chapter 2 PC-8001 Operation	7
2.1 Set-Up	7
2.2 Start-Up Without Disk Unit Attached	9
2.3 Start-Up With Disk Unid Attached	10
2.4 The Keyboard	12
 SECTION II — HARDWARE	 19
Chapter 1 Hardware Description	21
1.1 General Outline	21
1.2 External View (Rear)	23
1.3 PC-8001 Block Diagram	24
1.4 CPU Specifications	25
1.5 Memory	25
1.6 Input/Output Section	28
1.7 The Video Interface	28
1.8 Cassette Interface	30
1.9 The Keyboard	32
1.10 The Printer Interface	33
Chapter 2 Care and Service Advice	35
2.1 Cautions Concerning Power Source	35
2.2 Storage and Usage Conditions	35
2.3 Still More	36
2.4 Abnormalities	36

SECTION III — TIDBITS	37
Chapter 1 Examples of Simple Programs	39
Chapter 2 Program Storage on Cassette	47
Chapter 3 PC-8001 Terminal Mode	51
3.1 Terminal Mode Function Keys	52
3.2 Extension Interface RS-232C	53
Chapter 4 Explanation of Monitor	57
4.1 Executing the Monitor from BASIC	57
4.2 Command Types	57
4.3 Explanation of Commands	58
 SECTION IV — APPENDICES	 63
Appendix A Disk Files	65
Appendix B Machine Language Subroutines	75
Appendix C PC-8001 Memory Map	79
Appendix D Error Messages	81
Appendix E Character Code Chart	89
Appendix F Derived Functions	107
Appendix G Reserved Words	109

SECTION I
IN THE BEGINNING

CHAPTER 1

THE PC-8001 SYSTEM

The PC-8001 is a general purpose computer that is capable of performing a wide variety of tasks. It is capable of performing any calculation or scientific technique that could be expected of a "stand-alone" computer.

The host language of the PC-8001 is N-BASIC. The N-BASIC language of the PC-8001 has been created by Microsoft to fully utilize all of the special hardware available in the PC-8000 system.

1.1 PC-8000 System Components

Careful consideration has been given to the peripheral extensions to the PC-8001 computer. The available peripherals provide a gradual, proportionate extension to the basic system.

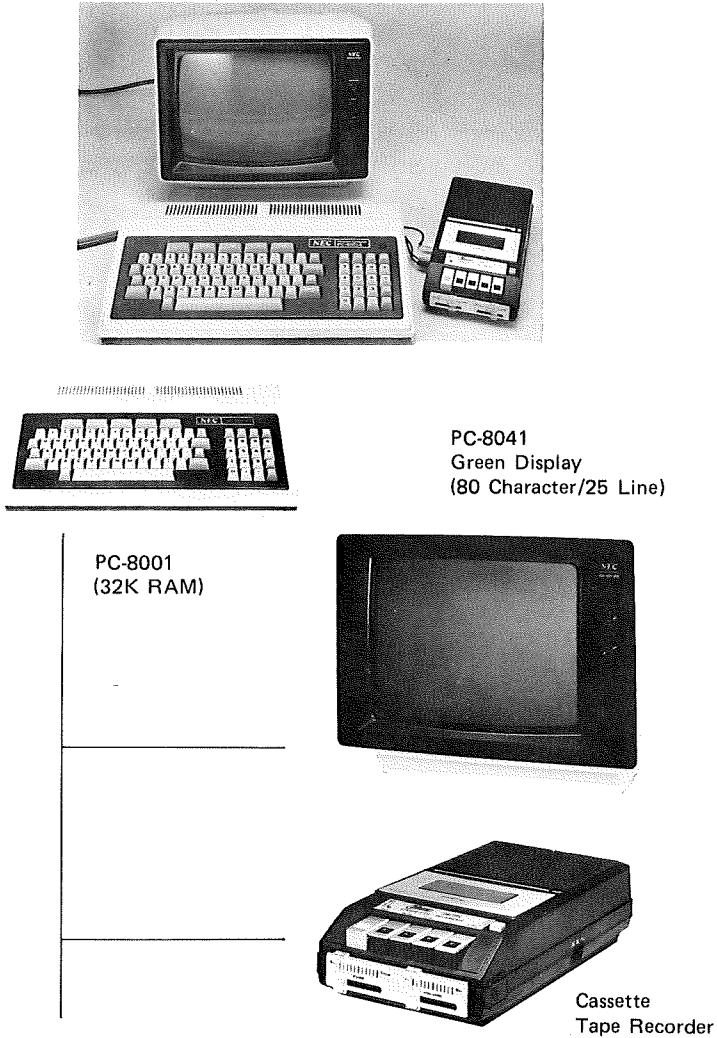


Figure 1 Basic System

1.2 PC-8000 System Peripherals

The following is a list of peripherals currently available for the PC-8001 Microcomputer. NEC will continue in its effort to supply quality peripheral products to add to the PC-8000 family.

Model	Description	Features
PC-8001	Computer	8-bit Computer, 32K RAM, N-BASIC in ROM, Printer Port, Serial Port, Cassette Interface, Color Output.
PC-8012	Expansion Unit	Comes with 32K RAM. Additional slots available for expansion boards.
PC-8023	132 Character Dot Matrix Printer	Bi-directional. Up to 132 columns. Fixed or proportional spacing. Block and dot graphics, 100 characters per second.
PC-8033	I/O Port for PC-8031	I/O Port necessary to connect PC-8031 to PC-8001 (not needed with 8012).
PC-8031	Dual Mini-Disk Unit	Intelligent Dual Mini Disk Unit, Single-sided, Double-density, 286K
PC-8032	Expansion Dual Mini-Disk Unit	Expands PC-8031 to four Drive 572K total storage.
PC-8041 (JB1201M)	12" Green Display	Green phosphor display, 80 character by 25 line, EIA video.
PC-8043 (JB1202DH)	12" Color Display	High Resolution Color Display, 80 Characters per line, R.G.B. signal (TTL) input.
PC-8062	RS-232 Cable	Serial cable that attaches to PC-8001.

CHAPTER 2

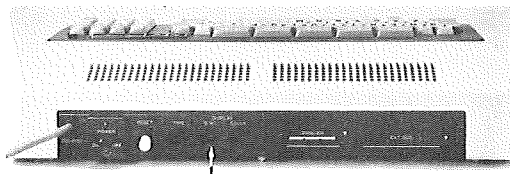
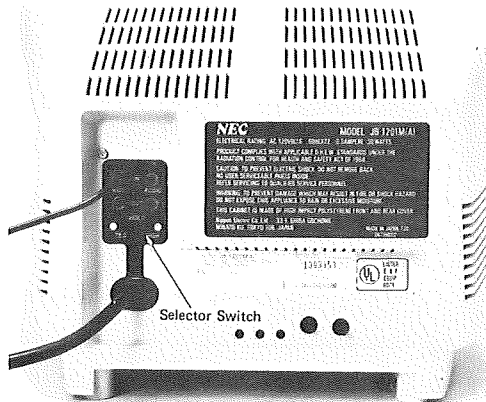
PC-8001 OPERATION

2.1 Set-Up

A description of the set-up of peripherals to the computer is given with the manuals supplied with those peripherals. The set-up of the PC-8041 Green Monitor and the Tape Recorder are described below.

Green Display (PC-8041) Hook-Up

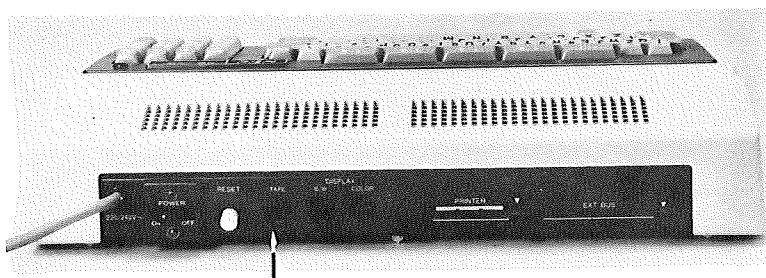
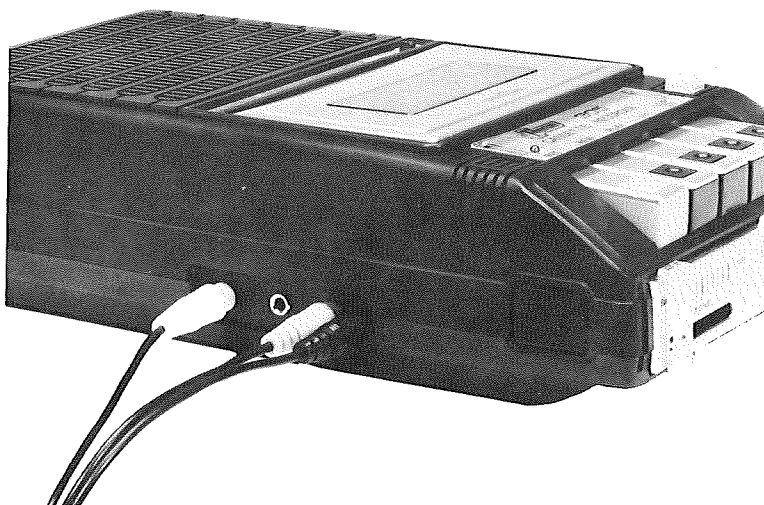
The green display is attached via its own cable (PC-8092). On the back of the display is a connector plate with a selector switch for video input. Set the switch to 75 Ω . Using the PC-8092 cable, connect the PC-8001 to the video in jack. For additional information and multiple monitor hook-up, consult the manual supplied with the PC-8041.



Cassette Tape Recorder Hook-Up

A cable is provided (PC-8093) to allow use of most cassette recorders with the PC-8001. One end of the cable has three connectors. Use these as follows:

Black	—	Remote
White	—	Earphone
Red	—	Microphone



2.2 Start-Up Without Disk Unit Attached

The PC-8001 computer plugs into a standard AC 115 V wall outlet. After the power switch is turned on, the green indicator on the PC-8001 will light. The following message will appear on the display.

NEC PC-8001 Basic Ver 1.1
Copyright 1979 (c) by Microsoft

OK

The above display will be in the 40 char by 20 line mode, and in Black and White. This is the normal power-on and reset mode.

Now enter this on the keyboard:

?fre(0)
26786
OK

The question mark (?) tells the computer to **PRINT** the following.

The reply, 26786, indicates the number of **RAM** bytes available to the user. While it is true that there is in fact 32768 bytes of **RAM** in the PC-8001, some of the **RAM** is used for display memory and workspace for N-BASIC.

Next enter the following:

?time \$
00:00:22
OK

Since the clock has not been set, the time displayed will actually be the time elapsed since power on. To set the correct time, type:

```
time $ = "10:30:40"   
OK  
?time $  
10:30:55  
OK
```

Now enter:

```
beep 1   
OK
```

This will cause the PC-8001 to produce a high pitch noise. To stop the beep, enter:

```
beep 0   
OK
```

With the above completed, a memory test is in order. This would be done under control of the monitor program. If, you desire to do the memory test at this time, consult Section III of this manual.

2.3 Start-Up With Disk Unit Attached

The PC-8031 reference manual contains the necessary instructions to connect the disk drives to your computer. Once you have done this, place the system disk into drive 1 and close the latch on the drive. Make sure the power switch on the disk drive is in the on position. Now turn the computer on (if it's on, press the reset button located in the back of the computer), and the Disk Operating System (DOS) will automatically be loaded into the computer from the disk. (We call this Booting the disk.) Without the DOS, the computer will not allow you to use the commands associated with the disk drives.

If the system loaded properly, your screen will look like this;

DISK VERSION

How many files (0-15)?

HT

auto

goto

list

run^cr

The computer is waiting for you to enter the number of files you expect to use at one time during this session. The files the computer is referencing are data files and not programs stored on the disk. We will review data files later in this chapter. For now, just press RETURN and your screen should say;

DISK VERSION

How many files (0-15)?

NEC PC-8001 BASIC Ver 1.1

Copyright 1979 (c) by Microsoft

OK

Ht

auto

goto

list

runCr

As we already know, the OK tells us that the computer is ready to accept our commands.

2.4 The Keyboard

The keyboard is much like a typewriter in that it has both upper and lower case characters, numerals and some special symbols. Push a key and the lower case character will be displayed. Push the **Shift** key, and a character key, and the upper case character will appear. For convenience, when typing in many contiguous upper case characters, the upper-case lock key is provided. If any key is pressed for longer than 1 second, it will automatically repeat.

The key assignments of the PC-8001 are shown in Figure 2. Pressing the **Graph** key enables the graphics characters shown in Figure 3. Pressing the **Alt Char** key enables the special characters and symbols, Figure 4.

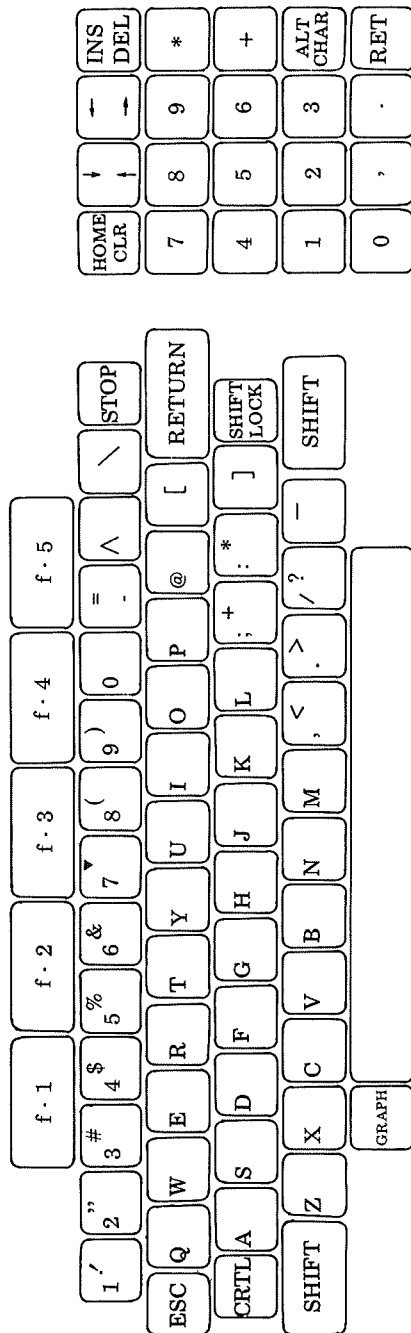


Figure 2 Key Assignments

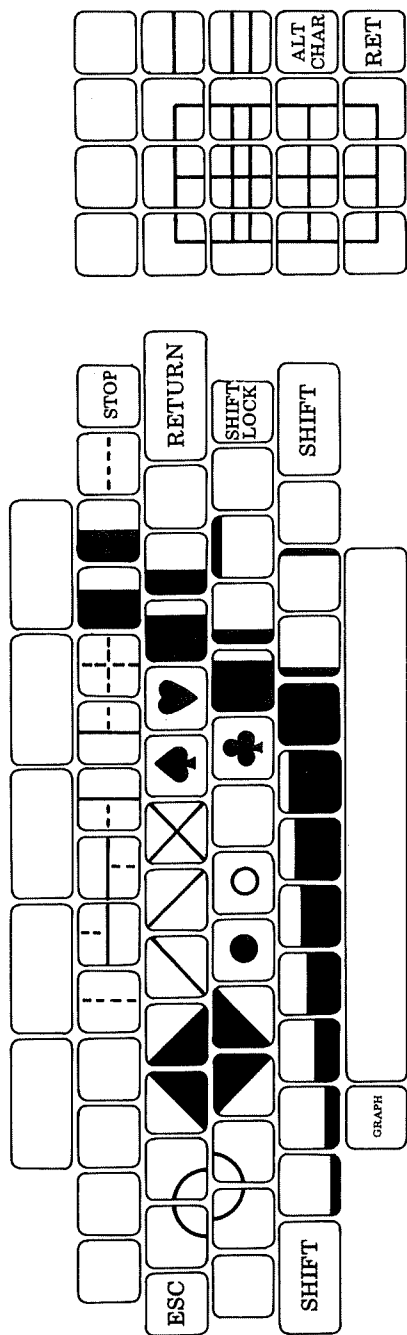


Figure 3 Graphic Characters

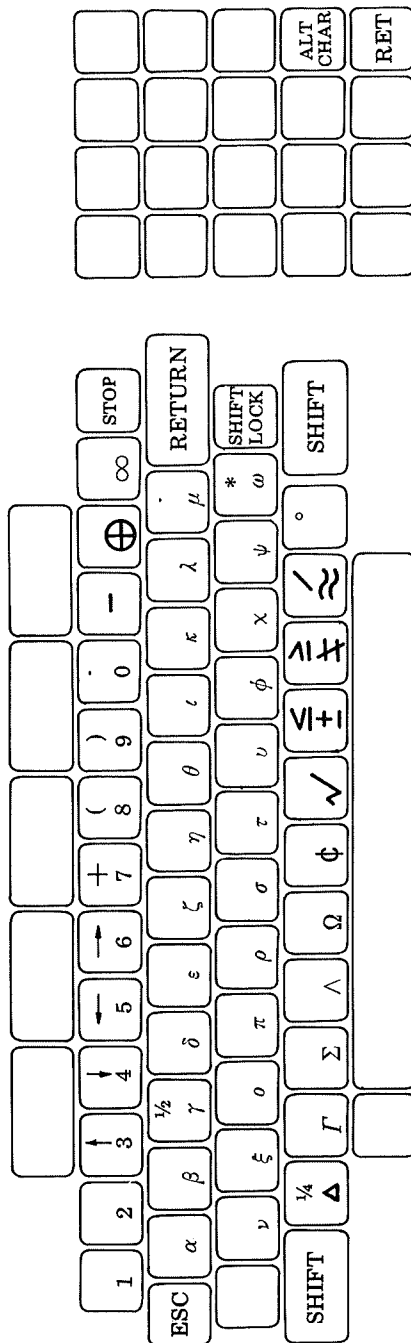






Figure 4 Special Characters and Symbols

Special Keys

1. **CTRL** — used with other keys for special functions.
2. **Shift** — used to display upper case characters and numerals and some symbols.
3. **Stop** — used to stop a BASIC program that is executing or listing.
4. **Ret** — return key — the same as |Return|
5. **Home
Clr** — clears the display screen. When the |Shift| key is pressed at the same time, the cursor moves to the Home position (upper left corner).
6.  — when pressed, this key moves the cursor up one line. When |Shift| is also pressed, this key moves the cursor down one line.
7.  — when pressed, the cursor moves right, one position. When |Shift| is also used, this key moves the cursor one position left.
8. **INS
DEL** — when pressed, characters are deleted from the cursor to the left. When combined with the |Shift| key, spaces will be inserted to the right of the cursor.
9.  to  — (Programmed function keys) Each function key has a function as listed below.

f.1	HT (horizontal tab)	Shift	+	f.1	time\$
f.2	auto	Shift	+	f.2	key
f.3	go to	Shift	+	f.3	print
f.4	list	Shift	+	f.4	list. ^{CR}
f.5	run ^{CR}	Shift	+	f.5	cont ^{CR}

The programmable function fields can display 6 characters each in the 36, 40 character mode and 12 characters in the 72, 80 mode.

To define a function key, use the following example:

key5, "run" + chr\$ (13) **Return**

This example defines key **f.5** as run + **Return**.

To list the function keys, type:

keylist **Return**

Example:

```
keylist
  H_          time$
  auto        key
  go to       print
  list        list. CR CR
  runCR       contCR
  Ok
```

10. **ESC** — pressing this key will cause a temporary halt in the running or listing of a program. To resume, press any key.

upper 4 bits

lower 4 bits

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D _E		0	@	P		p				∞	α	ϕ		
1	S _H	D ₁	!	1	A	Q	a	q			≥	3	ν	ω		
2	S _X	D ₂	"	2	B	R	b	r			·	γ	Δ	≈		
3	E _X	D ₃	#	3	C	S	c	s			*	4	β	✓		
4	E _T	D ₄	\$	4	D	T	d	t			≤	5	ξ	7		
5	E _Q	N _K	%	5	E	U	e	u			/	6	η	8		
6	A _K	S _N	&	6	F	V	f	v			·	ε	θ	9		
7	B _L	E _B	▼	7	G	W	g	w			↑	ρ	1	ι		
8	B _S	C _N	(8	H	X	h	x			½	σ	±	φ		
9	H _T	E _M)	9	I	Y	i	y			↓	ψ	υ	≠		
A	L _F	S _B	*	:	J	Z	j	z			←	Ω	π	χ		
B	H _M	E _C	+	;	K	[k	{			→	Γ	∧	°		
C	C _L	→	,	<	L	\	l	:			+	ο	2	ο		
D	C _R	←	—	=	M]	m	}			(δ	⊕	ζ		
E	S _O	↑	·	>	N	^	n	~)	κ	—	λ		
F	S _I	↓	/	?	O	—	ο				¼	Σ	τ	μ		

Figure 5 Character Code Chart

SECTION II
HARDWARE

CHAPTER 1

HARDWARE DESCRIPTION

1.1 General Outline

This section describes the hardware components of the PC-8001 personal computer. Primarily, the PC-8001 consists of a microprocessor, memory and input/output section.

The microprocessor is the heart of the PC-8001. This component is responsible for coordinating all activity in the system. The microprocessor is also responsible for executing the programs stored in the memory section. The PC-8001 uses the NEC μ PD780C-1 microprocessor chip which is compatible with the popular Z-80.

The memory part consists of two parts, the ROM and RAM. ROM stands for Read Only Memory. That is, this memory unit is shipped preprogrammed. Although you can read what is stored in ROM, you cannot use it to store programs that you wrote.

The second memory part in the PC-8001 is the Read/Write Memory (RAM). Because this is where you store programs that you write or load from tape or disk, this memory section is often referred to as user RAM (Random Access Memory).

Finally, we will discuss the Input/Output section. Because you will want to plug a wide variety of external components to your NEC PC-8001 (such as a cassette, printer or disk drives), much care has been taken to ensure that the PC-8001 will operate correctly with a wide range of external components or peripherals, as they are often called.

Figure 1 shows the PC-8001 system as you are familiar with it as well as what it looks like with the keyboard and cover removed. The computer itself consists of the power supply and the large printed circuit board mounted on the chassis of the case. The large board is called the mother

board because it contains the computer and all other peripherals plug into it.

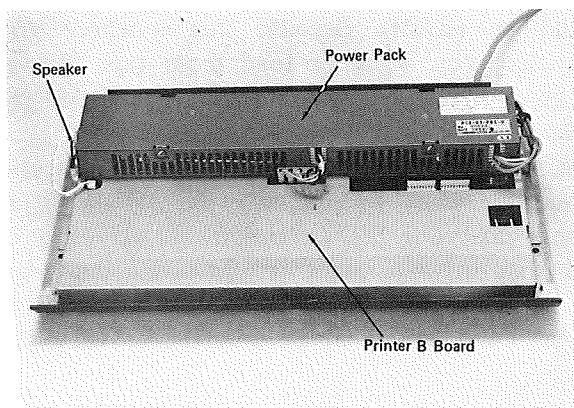
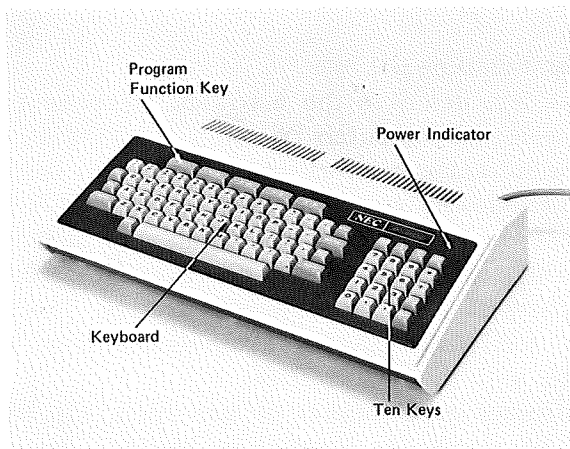


Figure 1 External and Internal View

These connections are shown in Figure 2. Notice that there is a power switch and reset switch on the back of the computer. In addition, we find the peripheral connections, namely, the cassette, monitor, printer and expansion ports.

1.2 External View (Rear)

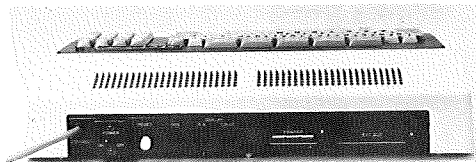


Figure 2 External View (Rear)

Figure 3 shows us a “block diagram” of the PC-8001 mother board (in the dotted box). For completeness, we also included the most popular peripherals on the block diagram.

These components of the NEC PC-8001 are described in further detail in subsequent sections.

1.3 PC-8001 Block Diagram

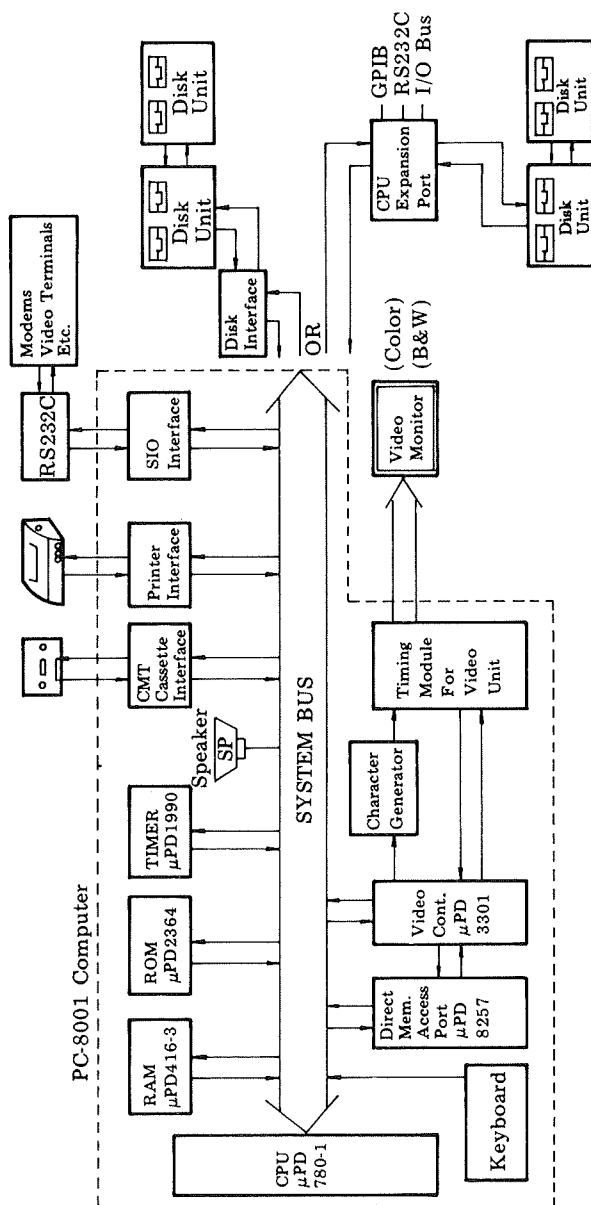


Figure 3 PC-8001 Block Diagram

1.4 CPU Specifications

As we stated earlier, the PC-8001 is based on the μ PC-780C-1 microprocessor. (This microprocessor is often referred to as the **central processing unit (CPU)** as this is where the computer does its “thinking”. The CPU operates on a 3.9936 MHz — 4 MHz (millions of cycles per second) clock derived from a 15.9744 MHz quartz oscillator.

The μ PC780C-1 is an 8-bit microprocessor. Data is represented 8 bits at a time and this is conveniently referred to as a byte (half of a byte is, you guessed it, a nibble).

The Z-80 can communicate with the memory and I/O sections over a “bus”, so called because information is bussed on a group of wires. The bus has three major sub-components. The data bus, over which information is transferred, the address bus, over which information pertaining to where the data is to go is stored, and the control bus, which is used to coordinate the memory and I/O sections.

Three important control signals are **RESET**, **WAIT** and **INTERRUPT**. When you press the **RESET** button, you are telling the computer to unconditionally stop whatever it is doing and start all over from the beginning. This process is performed automatically when you turn power on.

The **WAIT** control signal allows a system component to hold up processing until the component can complete what it was asked to do. For example, if we have a memory that cannot operate at CPU speed, the memory can ask the CPU to wait while it accesses the data requested by the CPU.

Also, we have an interrupt signal which is a method of tapping the CPU on the shoulder to get its immediate attention.

1.5 Memory

As stated in the general outline, there are two types of memories, the **ROM** and the **RAM**.

(1) ROM

The ROM section of the PC-8001 has been previously programmed and cannot be reprogrammed. The computer uses the programming in ROM to allow the user to write, edit and run programs, take or store programs from other devices such as disk or tape units and implement special functions.

When the PC-8001 is first turned on the computer automatically resets, runs the program in ROM, and waits for user or peripheral interaction. The computer also automatically returns to the ROM programming after the "end" of a program.

The standard PC-8001 comes with 24K bytes of ROM which holds the program described. The ROM section can be expanded 32K bytes for larger or more complex needs.

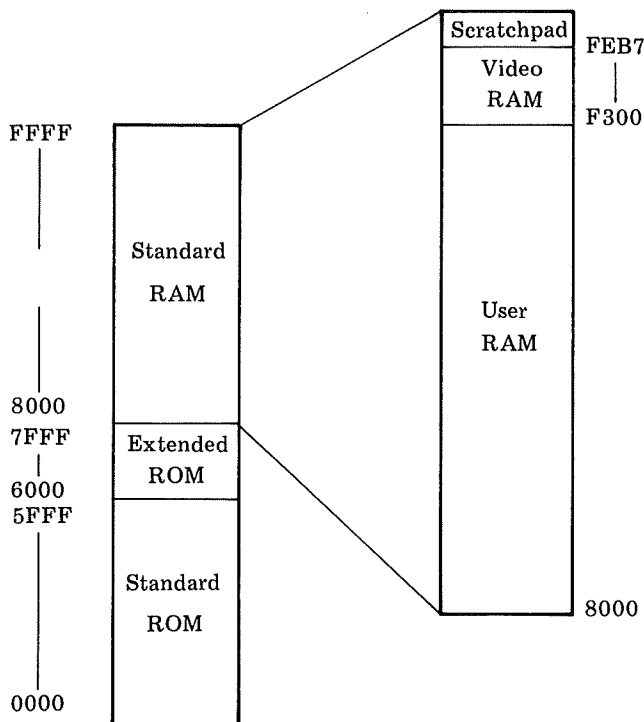


Figure 4 PC-8001 Memory Map

(2) RAM

The RAM section of the PC-8001 is the area provided for running or temporarily storing user programs.

Since the RAM can be changed, any program using RAM to store data or other information must be carefully written as to not change any of the RAM area containing the program.

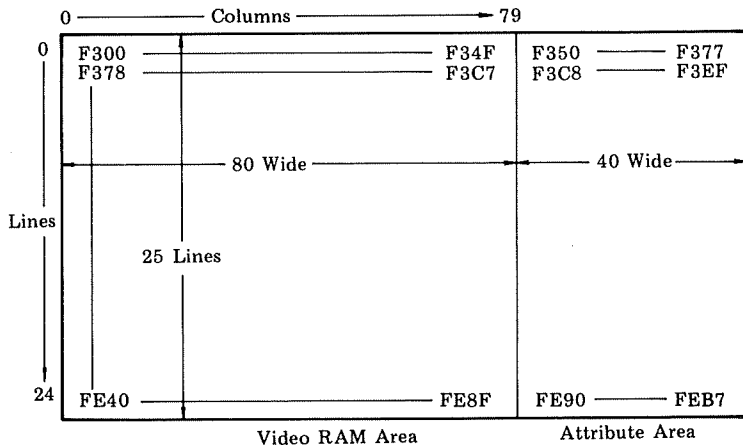
The RAM section of the PC-8001 is a temporary storage medium. It can be changed and when power is turned off, the RAM loses its contents.

The standard PC-8001 contains 32K bytes of RAM.

The memory map for the PC-8001 is shown in Figure 4. The addresses are shown in hexadecimal from 0000 to FFFF which is a total addressable space of 64K bytes.

The memory space from 0000 to 5FFF is the standard 24K bytes of ROM. The memory space from 6000 to 7FFF allows for the expansion of ROM to 32K.

The standard RAM is further broken down into the user RAM area and an area used for the storage of characters for the video monitor (from F300 to FEB7). The memory area FEB8 to FFFF is reserved for workspace for scratch-pad or user memory.



Video RAM Composition

1.6 Input/Output Section

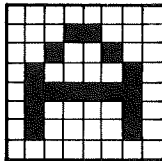
The input/output (I/O) section of the PC-8001 is used to interface, or connect, external devices to the computer. These external devices, also called peripherals because they are outside of the computer, perform two types of functions. Some peripherals store data such as disks and cassettes. Another group of peripherals process the data so that you, the user, can understand it. This group of peripherals includes the CRT, keyboard and printing devices.

The PC-8001 will directly connect to a monitor, a cassette player, printer, terminals and an audio speaker. The capability has been provided to add an optional Floppy Disc Drive System (with up to four drives) and an optional I/O Expansion Interface.

We will describe the video interface first since this unit ties in closely with the description of memory discussed in the previous section.

1.7 The Video Interface

Essentially, the video interface stores the information put on the monitor or TV in random access memory. If you look closely at the video display, you will notice that the characters and images are composed of small dots of light. You can imagine that each one of these dots is stored in a memory location. But how much memory is required to store a screen of output is dependent on the number of characters we wish to display. Each character is composed of 64 dots.



The letter "A" stored in a dot matrix.

In the PC-8001, about 3K bytes (1K = 1024 bytes) are used to store video information. Now, if we work out the numbers, we would find that if we needed 64 bits to store one character, we would need 8 bytes per character (because there are 8 bits in a byte). Well, in the 80 column mode, that corresponds to 80×64 bits per line (or 640 bytes). And since we have 25 lines, we would need 16000 bytes to store 1 screen of output. In other words, we would have to use all of the PC-8001's user RAM to store output and we would have no room for programs. Well, as it turns out, we can make much more efficient use of our memory space. Since characters do not change shape, (i.e., an a always looks like an A), we can permanently store each character in a Read Only Memory (ROM).

Now, when we want to display a character, we simply get its address in ROM and send it out to the CRT controller. So we don't actually store the character, but rather we store the address of where the character is stored in the character generator ROM. What this means is that we can now store 192 characters and 56 special graphics characters (for a total of 248) in 2048 bytes of ROM ($64 \text{ bits/character} \times 256 \text{ characters/8 bits/byte}$). By the way, we say 256 characters although we need only 248. Well now, the video character generator ROM uses 2048 (2K) bytes and the video RAM requires 3K for a total of only 5K bytes. This is a significant improvement in RAM utilization (over the 16K we calculated earlier).

We have seen that the video RAM is used to store information pertaining to what characters to display. But, if we can store 80×25 characters (2000 characters) on a screen, we need only 2K to represent a screen of data and we stated that the video RAM needed 3K. Where did the other 1K go? Well, this section of video RAM is where we store attributes, that is the distinctive features of a particular character (such as blinking, reverse video and color). You can imagine this as a building block approach, a character is retrieved from the video character generator

and its attributes are retrieved from video RAM. The CRT controller puts the 2 parts together and voila! A Character.

Now you might be wondering how the PC-8001 can generate characters and put them on the screen and still have time to run your program. The answer is that it doesn't. The PC-8001 uses an extremely efficient scheme for generating its video screen known as direct memory access (DMA). This technique allows the CRT controller to access memory and update the video screen without the CPU performing any of the tasks involved in this process. While the Z-80 CPU is busy, the DMA controller sneaks into RAM and pulls out a character. This actually happens everytime an instruction cycle is executed by the processor. Since the DMA controller and the Z-80 CPU access memory in a shared scheme, both devices can make very efficient use of their time; the Z-80 can spend its time running your program and the CRT controller can relieve the CPU from the task of outputting video screens. This technique allows the PC-8001 to operate more efficiently (and faster) than other processors because it is not wasting time performing the task of driving the video screens.

1.8 Cassette Interface

The cassette interface allows the PC-8001 to store data on an ordinary cassette player and retrieve the information at a later time. This poses an interesting problem in that the cassette stores audio information while the computer stores digital information. The solution, then, is to translate the computers digital language into a series of tones that can be recorded on tape. Now, since all information that is stored in the computer can be represented by either a zero or a one (based on the binary number system), it appears that we need two tones to represent digital information on tape. This is exactly the case and we now go about picking two tones to represent out 2 binary states. These tones sound like "Whistles" if you listen to them since their frequencies are close to those you generate when you pucker up and whistle. Now, what we do is let the

interface output a low tone (1200 Hz) to correspond to one binary state and a high tone (2400 Hz) to correspond to the other state (the digits 0 and 1 can be thought of as states and a bit inside the computer can be in one of either state at a time, no other states are valid).

So here we are outputting a constant whistle until the data changes state (from 0 to 1 or from 1 to 0). What we must do is shift frequency to get to the correct state. Thus, if we are at 1200 Hz* (low whistle), we must shift frequency to the other state 2400 Hz (high whistle).

*Hz = Hertz and is a measure of how many times an event repeats itself in time (cycles per second) a second hand on a clock has a frequency of 1/60 Hz since it takes 60 seconds to complete 1 cycle on the face of a clock).

This technique of storing binary data on tape is called frequency shift keying (**FSK**) since all we need to do is change from one frequency to another to represent our data.

Similarly, to read data back, we just translate the frequencies on tape to ones and zeros within the computer.

The system tends to be very reliable since very sensitive circuits exist for translating voltages (1's and 0's) to frequencies (1200 Hz to 2400 Hz). The transmission of this data from computer to tape and back again occurs at a rate of 600 bits per second (it would take just over 3 minutes to store all 16K of user RAM on the cassette). The unit we refer to as bits per second (bps) also is named the baud (for Mr. Baudot who did early work in the field of communications).

The cassette I/O also contains a remote start-stop mechanism to control the cassette recorder under software control. To use this I/O with your tape recorder, you will need to use the cable included with your system. Notice that one end of the cable contains a round plug with 8 pins in it. This is called a DIN connector. This plugs into the

back of the PC-8001 in the matching DIN socket on the left side of the computer as you look at the back of the unit. You will only be able to plug it in one way, so don't worry about causing any harm, you won't.

Connect the red plug at the other end of the cable into the "mike" input on the cassette (also called "mic"). This is equivalent to plugging a microphone into the cassette only now the computer is going to do the talking.

Connect the white connector to the earphone jack on the recorder. This allows the computer to listen to the whistles previously recorded on the tape.

If your cassette has a remote command option, plug the black connector into the Remote + (REM +) jack and the red plug into the Remote (-) (REM -) jack. This will enable your cassette to be started and stopped by the computer.

Now, you should put the volume control at 5 (about halfway between 0 and full volume). The tone control, if your cassette is so equipped, should be placed at its highest position (10 or high). You are now ready to use the cassette interface. Refer to the procedures section of the manual to see how the interface may be used to load and store programs.

1.9. The Keyboard

The PC-8001 keyboard interface is conceptually very simple. The CPU "scans" the keys periodically to see if a key has been depressed. If a key has been depressed, the keyboard will have a "flag" raised indicating to the CPU that it wants to be serviced. The CPU lowers the flag indicating that it is servicing the request. Then it simply scans the keys until it finds the key that was depressed. This key has associated with it a code that will allow the video interface to use the key code to generate the necessary character.

1.10 The Printer Interface

Another useful peripheral to include in your system is a printer to provide a copy of your programs and data on paper (hard copy, as opposed to "soft copy" such as the video display).

The printer interface that we adopted for use by the PC-8001 is identical in operation to the interface used by Centronics in their line of printers (in fact, this is a very common interface and a wide variety of printer manufacturers also use it).

This interface is "parallel". That is, it transfers 8 bits at a time along 8 individual wires. That is in contrast to a serial interface, such as the cassette I/O, which only transfer 1 bit at a time.

Now if you look at the connecting cable, you will notice that it contains more than 8 connectors so the computer and printer must share more than just 8 data bits. This is in fact the case. The additional lines control the flow of information between the computer and the printer. These lines are called "handshake" signals because they allow the printer and computer to shake hands when they agree on a transfer. Let's take a look at how this works.

Imagine if you will that the computer has been programmed to print your name and phone number on the paper inserted into the printer. If your name is "Jill" and your phone number is "377-2177" the computer must ask for 13 characters to be printed: name (4) and phone number (8) separated by a blank space (1). Now the first character is put on the 8 data lines (remember characters are represented in an 8 bit code). Before the CPU can send the data down the line, it must check the printer's ready line. In other words, the CPU asks the printer if it is ready. If the printer is not ready, as when it is busy printing another character, the CPU must wait. Once the printer is ready, the CPU sends the character down the line. But how does the printer know that a character is being sent? Well, simultaneous with sending the character, the CPU "shakes" its strobe line. The printer now knows that a character is

coming in. So it sends an "acknowledge" signal back to CPU saying yes, I got the character. Now we have printed one character, "J" in this case, and we are ready to print the next character (the "I" in Jill's name).

This continues until all characters are transmitted.

CHAPTER 2

CARE AND SERVICE ADVICE

2.1 Cautions Concerning Power Source

- 1) After turning power switch on, please wait 5 seconds before entering. Similarly, after entering, please wait at least 5 seconds before pulling out the power plug.
- 2) The power source must use AC 115 V (60 Hz).
- 3) When pulling out the power cord, please grasp near the plug.

2.2 Storage and Usage Conditions

- 1) To protect the machine temperature from rising, avoid use in hot places and keep the case open for ventilation. Avoid use as well as storage in places of severe humidity changes and locations of extremely hot or cold temperatures.
- 2) Avoid using or storing in the direct rays of the sun and near heaters.
- 3) Do not use or store in places of severe dust or dampness.
- 4) As the machine is of precision-made electronic parts, avoid violent shocks or vibration while in use or in storage.
- 5) Be careful not to let water or any liquids into the interior of the machine as it could come in contact with the metal parts during use.
- 6) Chemicals should not come in contact with this machine and chemicals should not be used in the air around it.
- 7) Do not store or use machine partially dismantled, as this is the source of accidents and electric shock.
- 8) Do not pile or stack things on top of the machine.
- 9) Do not use near radios or televisions as interference will occur on them. Correspondingly, the strong magnetic field produced by radio and TV will

adversely effect the machine. Keep the PC-8001 separate from radios and TV's.

2.3 Still More

- 1) The cabinet may be cleaned with a soft cloth dampened with water or with a mild cleaner. When benzine, paint thinner, chemicals or any volatile substance is used, there is risk of disformity and discoloration. Also, use caution with insecticides, as the same risk is present.

2.4 Abnormalities

- 1) If the machine gives off a strange odor or is overheating, pull the power plug immediately and consult the store where the PC-8001 was purchased.

SECTION III

TIDBITS

CHAPTER 1

EXAMPLES OF SIMPLE PROGRAMS

This chapter will show basic techniques of programming in N-BASIC by use of some examples. With the exception of program 5, all the programs are made as simple as possible and it is hoped that the results will prove to be of great help and interest. The following is a list of the example programs.

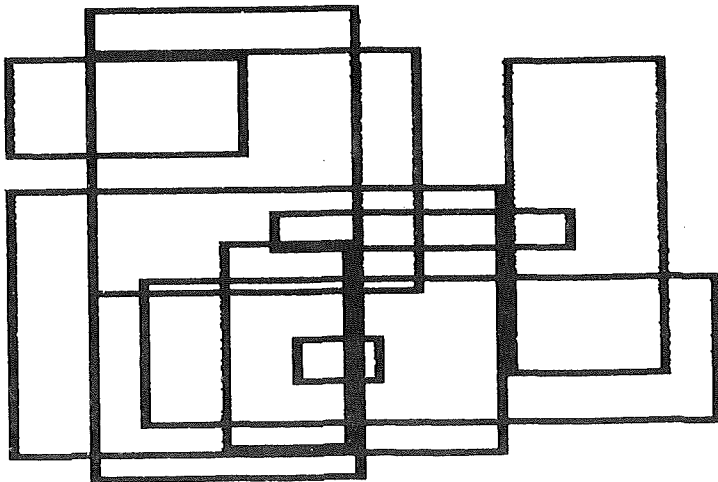
- Program (1) Random box drawing program. Learn how to use the **RND** statement.
- Program (2) An interesting plot program. By inserting sine and cosine functions, the basic operation of the **PSET** statement can be understood.
- Program (3) Note this interesting moving program.
- Program (4) The N-BASIC commands **TIME\$**, **RIGHT\$**, and **VAL** are demonstrated here. The display is cut up into 60 second intervals.
- Program (5) This is a 3-dimensional plot program. It takes a long time to run however.

PROGRAM (1)

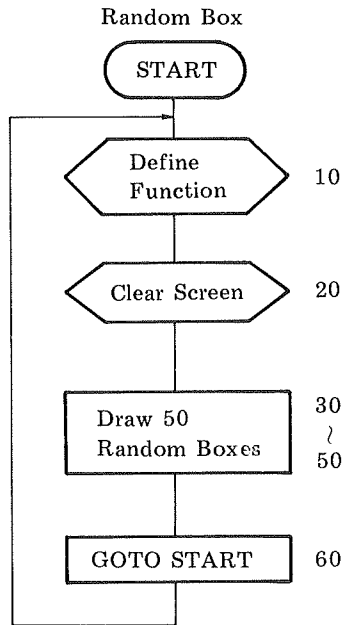
list

```
10 CONSOLE 0, 25, 0, 1
20 WIDTH 80, 25
30 DEFFNR (R)=RND(1)*R
40 COLOR FNR(7)+1,,1
50 PRINTCHR$(12)
60 FOR I=1 TO 50
70 LINE (FNR(159),FNR(99))-(FNR(159),FNR(99)),
  PSET,B
80 NEXT
90 RUN
OK
```

Program (1) when RUN



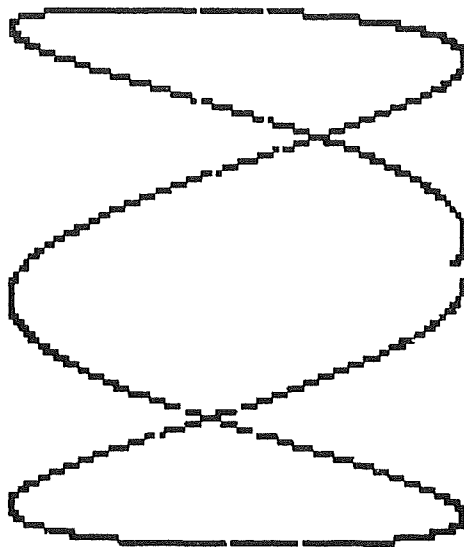
PROGRAM(1) FLOW CHART



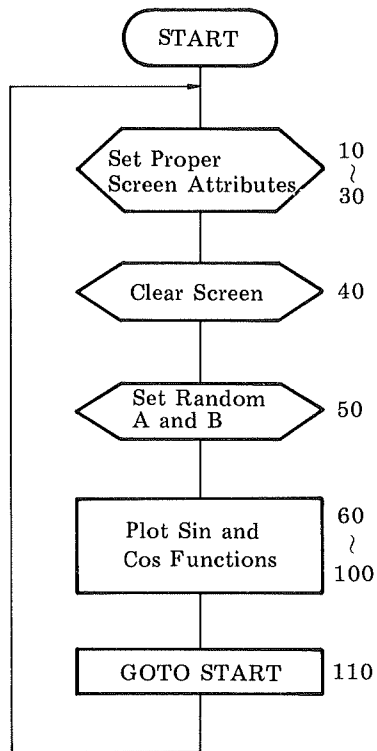
PROGRAM(2)

```
Ok
list
10  CONSOLE0,25,1,0
20  WIDTH80,25
30  COLOR0,0,1
40  PRINTCHR$(12)
50  A=RND(1)*5:B=RND(1)*5
60  FOR I=1 TO 800
70  Y=SIN(I/100*A)*45+46
80  X=COS(I/100*B)*45+80
90  PSET(X,Y)
100 NEXT
110 GOTO 10
Ok
■
```

Program(2) when RUN



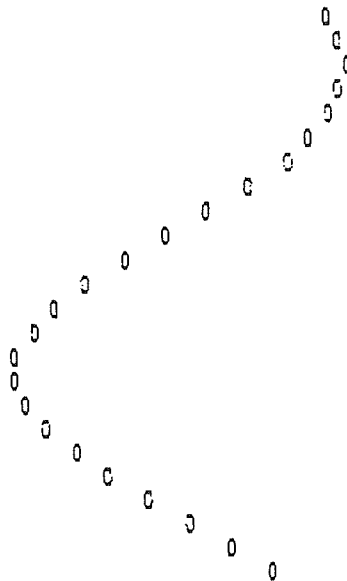
PROGRAM(2) FLOW CHART



PROGRAM(3)

```
Ok
list
10 WIDTH80,25
20 P=INT(80*RND(1))
30 V=(40-P)*.05
40 PRINTTAB(P)"O"
50 P=P+V
60 V=V+(40-P)*.06
70 V=.97*V
80 IF RND(1)<.04 THEN V=V+SGN(V)
90 GOTO 40
100 RUN
Ok
|
```

Program(3) when RUN

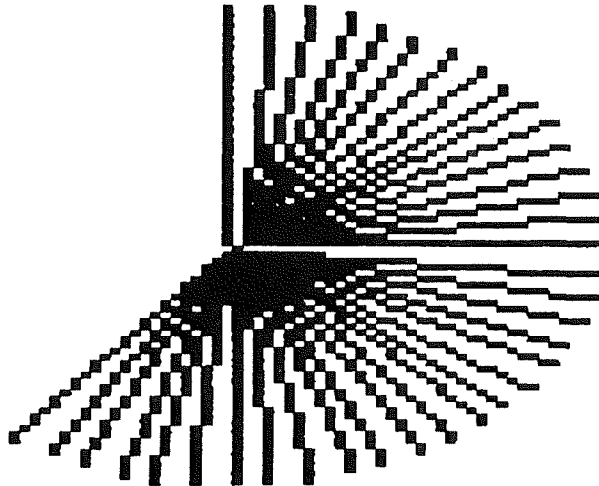


PROGRAM (4)

```
list
10 COLOR,,1:WIDTH36,25
20 PRINTCHR$(12)
30 C=6.28/60:D=6.28/60
40 J=VAL(RIGHT$(TIME$,2))+45
50 Y=SIN(J*C)*40+45
60 X=COS(J*D)*36+36
65 BEEP1:FOR Z=1 TO50:NEXT:BEEP0
70 LINE(X,Y)-(36,45)PSET
80 IF T$=TIME$ THEN 80 ELSE LOCATE
90 0:PRINTTIME$
90 T$=TIME$
100 IF RIGHT$(TIME$,2)="00"THEN10 EL
SE 110
110 GOTO 40
120 GOTO 10
OK
```

Program(4) when RUN

02:31:36



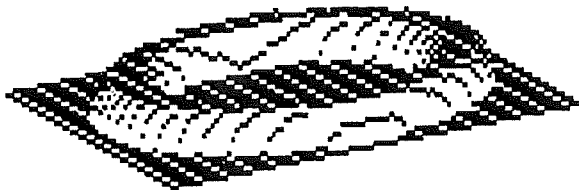
PROGRAM(5)

```

5 DIM M1(255),M2(255)
10 CONSOLE,,,0,1:WIDTH 80,25
20 COLOR 7,,1:PRINT CHR$(12)
30 R1=RND(1)*90:IF R1<70 THEN 38
40 R=RND(1)*25:IF R<10 THEN 48
50 LOCATE 15,1:PRINT "3 DIMENSION"
60 LOCATE 13,2:PRINT "GRAPHIC DISPLAY D
EMONSTRATION"
70 COLOR 6
80 R1=INT(R1):R=INT(R)
110 FOR I=8 TO 255:M1(I)=191:M2(I)=8:NE
XT I
120 COLOR (RND(1)*7+1)
130 B1=59:B2=100:S1=12
140 FOR Y=-100 TO 100 STEP 10
150 B1=B1-2:B2=B2-1:H1=B1:V1=B2
160 FOR X=-100 TO 100 STEP 2
170 H1=H1+1:IF X/S1=INT(X/S1) THEN V1=V
1-1
180 IF X*X+Y*Y=0 THEN Z=0:GOTO 220
190 T=(RND(1)-(X/R1)/SQR(X*X+Y*Y)-X)^2-(Y/R
1/SQR(X*X+Y*Y)-Y)^2)
200 IF T<0 THEN T=0
210 Z=SQR(T)/2
220 H=H1:V=V1-Z
230 IF V<=M2(H) THEN IF V>=M1(H) THEN 2
70
240 IF V>=M2(H) THEN M2(H)=V
250 IF V<=M1(H) THEN M1(H)=V
260 PSET (H,V)
270 NEXT X:NEXT Y
280 BEEP1:FOR Z=1 TO 1000:NEXT :BEEP0:E
ND

```

3 DIMENSION
GRAPHIC DISPLAY DEMONSTRATION



CHAPTER 2

PROGRAM STORAGE ON CASSETTE

Cassette tape can be used with the PC-8001 for two basic functions:

- (1) Program storage
- (2) Data storage

With the PC-8001, the program and the data that resides in memory is completely erased when the power source is cut off. (The N-BASIC interpreter itself is not erased). Consequently, programs and data used more than once can be stored on cassette tape. These programs can then be loaded directly into memory when you wish to use it — avoiding the trouble of entering the program from the keyboard.

Program loading from cassette tape uses the **CLOAD** instruction as explained below.

CLOAD

Format:

CLOAD <filename>
CLOAD? <filename>

Purpose:

To load a program from cassette tape into memory.

To verify a program in memory with that on tape.

Remarks:

CLOAD executes a **NEW** command before loading the program. <filename> is a string of up to 6 characters that was specified as the filename when the program was saved using the **CSAVE** command. <filename> must be enclosed in quotes.

CLOAD? verifies tapes by comparing the program currently in memory with the file on the tape having the same filename. If they are the same, **N-BASIC** prints "OK". If not, **N-BASIC** prints "BAD". The program in memory is not altered when using this instruction.

Before a **CLOAD** command is executed, make sure that the tape recorder is properly connected and in the play mode, and that the tape is properly positioned. If there are other programs on the tape before the one you wish to load, the words "SKIP: <File>" will appear, where <file> is the name of the program being skipped. When the file you wish to load is found, the words "FOUND: <Filename>" is displayed, where <Filename> is the name of the program you used in your **CLOAD** instruction. After the program is found, a flashing asterisk will appear on the screen while the program is being loaded.

N-BASIC always returns to the command level after execution of a **CLOAD** instruction.

Example: **CLOAD "MAX2"** Loads file "MAX2" into memory.

CLOAD? "MAX2" Compares file "MAX2" on tape with program currently in memory.

To save a program onto cassette tape, the **CSAVE** instruction is used as explained below:

CSAVE

Format: CSAVE <string expression>

Purpose: To save a program presently in memory to cassette tape.

Remarks: Programs saved on tape are identified by a file name. When the statement **CSAVE** <string expression> is executed, N-BASIC stores the program currently in memory on cassette tape, using the first six letters of <string expression> as the file name. More than six letters may be specified but only the first six will be used. <file-name> must be enclosed in quotes. **CSAVE** may be used either as a program statement or a direct mode command.

Example: CSAVE "TIMER" Saves the program currently in memory under the file name "TIMER".

To save or input data from tape, see the **PRINT #** and the **INPUT #** commands described in the **SOFTWARE** Section of this manual.

CHAPTER 3

PC-8001 TERMINAL MODE

The PC-8001 isn't just a personal computer, as it can also be used as a terminal to other computers. In this terminal mode, the hook-up to the other computer is through an RS-232C Interface. If this interface is not connected and the **TERM** command is executed, the computer will show an asterisk (*). This shows that an error has occurred. An explanation of the **TERM** command is as follows:

TERM

Format:	TERM <word length>, <parity specification>, <clock drive ratio>, <auto-line feed specification>
Purpose:	To put the PC-8001 in the terminal mode.
Remarks:	In the terminal mode, the PC-8001 can communicate with other equipment via an RS-232 interface. The terminal mode is entered from the BASIC mode by entering the TERM command. <word length> can be specified as "a" meaning that an 8-bit word, such as in ASCII code, will be used, or as "j" meaning that a 7-bit word will be used. <parity specification> can be 0, 1, or 2 corresponding to: no parity, odd parity, and even parity, respectively. <clock drive ratio> can be either 0 or 1, corresponding to divisions of 64 and 16. The final parameter, <auto-line feed specification> can be specified as 1, meaning that the terminal will automatically give one line feed upon receiving a carriage return, or

as 0, meaning that no auto-line feed is given.

Example: **TERM a,0,1,1**

The PC-8001, when in the terminal mode has 2 basic actions:

- 1) characters entered from the RS-232C channel are displayed in the CRT
- 2) characters entered from the keyboard are sent through the RS-232C channel

3.1 Terminal Mode Function Keys

In the terminal mode, the function keys 6-10 have the following special meanings:

- 6: activates or deactivates expression of control code
- 7: full or half duplex
- 8: activates/deactivates echo in LP
- 9: page print to LP
- 10: sends line feed to printer

Pressing keys 6-8 will produce a certain situation. Pressing them again will toggle them back. The default for keys 6-8 when the **TERM** command is first executed are:

- 6: control code not activated
- 7: full duplex
- 8: no echo to printer

To return to the N-BASIC operating system from the **TERMi**nal mode, use control B (press the control key while pressing B). Entering the **TERMi**nal mode will not change the N-BASIC program you currently have in memory.

3.2 Extension Interface RS-232C

*****CAUTION*****

This section will tell how the PC-8001's attached "extension interface" is applied. That is, when the PC-8001 is in use, there are certain statements under which it cannot be utilized.

The extension interface RS-232C uses a "cut in". During BASIC programming, RS-232C data can be entered and the buffer put aside. This feature means that BASIC phrases entered through the RS-232C have the same meaning as those entered through the buffer, and in actuality, entered by the buffer!

1. RS-232C Port Early Establishment

The condition of the RS-232C port just after power is on is uncertain. Early establishment of the RS-232C port before using is a necessity. For this, use the following INIT phrase:

INIT% <port number>, < μ PD8251 mode byte>,
< μ PD8251 command byte>

The port number is (1) one or 2 (two). (The extension interface has a 2ch RS-232C interface.

The RS-232C data usage components, namely data length, parity, baud rate factor, etc., designate by μ PD8251 mode byte, and command byte. This feature, when the operator wishes to use the RS-232C channel, requires a complete understanding of the μ PD8251 and RS-232C standards. Therefore, if you experience a crisis and don't understand the standards, it would be best to know these from the beginning. Please study the following material.

EIA STANDARD RS-232C
Electronic Industries Assoc.
μPD8251 Manual

2. RS-232C Port Output

Use PRINT% as a special PRINT phrase model type.

PRINT% <port number>, <list of produced data>

(Sample) PRINT%, 1,A,B,C,D

Will produce the values of variable A,B,C,D in RS-232C # 1 ch.

PRINT% 2, "EXAM",A\$

Will give value of A\$ variable and character constants in RS-232C # 2ch.

3. RS-232C Entry from Port

As stated before;actually entry is from buffer.

INPUT% <port number>, <variable list>

From inside the buffer, the entered character samples are seen as CR, LF characters, and are substituted for the designated variables. If these characters are not in the buffer, the entered ones will pass through the RS-232C.

4. Limits of known characters entered in buffer presently.

PORT <port number>

(Sample) A=PORT(1)

Will substitute length of characters entered in the RS-232C #1 ch buffer in A.

5. Limit on bringing designated length character samples from RS-232C buffer.

INPUT\$(**<brought character length>,%<port number>**)

(Sample) A\$=INPUT\$(10,%2)

Substitutes the lead 10 characters of the RS-232C #2 ch buffer in A\$.

6. Concerning Buffer Length

The RS-232C buffer has a length of 127 character spaces in each channel. Should there be an overflow of entries from the buffer of the RS-232C port, the following error will occur:

“Communication buffer overflow”

CHAPTER 4

EXPLANATION OF MONITOR

4.1 Executing the Monitor from BASIC

Using the N-BASIC **MON** command, the monitor can be actuated. When **MON** **Return** is entered, an asterisk (*) will appear indicating a situation where monitor command entry is possible.

```
OK
MON RETURN
*
```

4.2 Command Types

In the monitor, there are 8 types of commands as listed below. The next section (4.3) will explain the contents in detail.

Command	Contents	Command	Contents
S	Set Memory	W	Write Tape
D	Display Memory	G	Jump
L	Load Tape	CTRL-B	Return Basic
LV	Load Verify Tape	TM	Test Memory

4.3 Explanation of Commands

(1) S(Set Memory)

This command used when changing memory contents.

*Sxxxx Return (xxxx=address)
after Key, "S" enters address change.

As for the addresses, it doesn't matter how many lines are entered, but if over 4 lines, the lowest four(4) ranking lines become significant.

* S 1 2 3 4 C 0 0 0

RETURN

C 0 0 0 0 0 —

if *Sxxxx Return is entered, the indicated address contents becomes the KEY entry.

(Entry possibility KEY)

Change data. 2 column HEX code (00-FF)

Subtract 1 address H(CTRL-H)

end of procedure Return or Stop key

indicate data from next address (space)

When a KEY is entered, and there is an error, a question mark (?) will be indicated nearby and awaits an entry command.

When there is no entry error or the **Return** key has already been entered, the address process will be able to continue since the last address is set in memory.

```
* S C 0 0 0 Return
C 0 0 0 0 0 -12 0 1 - Return
* S Return
C 0 0 1 0 1 -34 0 2 - G
?
* S Return
C 0 0 2 0 2 -
```

Note: Use caution when entering the **STOP KEY** during the process as the process is not set in the memory counter.

(2) D(Display Memory)

*Dxxxx, yyyy **Return** (xxxx=Start address, yyyy=end address)

This command can be used when displaying memory contents. “D” dumps the memory from the first to the last address entered. Once again, for the address, it does not matter how many units are entered, but if more than 4, then only the lowest ranking 4 are significant.

```
* D C 0 0 0 , C 0 0 F Return
C 0 0 0 0 0 0 1 0 2 0 3 0 4 0 5 0 6 0 7
C 0 0 8 0 8 0 9 0 A 0 B 0 C 0 D 0 E 0 F
*
```

If only a starting address is specified, 16 bytes will be displayed.

***DC000** **Return**

would become the same as the previous sample.

Next, when you wish the flow of data to stop, enter **Esc** key, and it will temporarily halt. When the operator wishes for the listing to continue, he simply enters the **Esc** key once more. If the operator wishes to discontinue the listing, he should enter the **Stop** key and the output will stop.

(3) W(Write Tape)

This command is used when recording (writing) memory contents on tape (audio cassette).

***Wxxxx, yyyy** **Return** (xxxx=start address,
yyyy=end address)

After Key, W enters the first and last addresses of the memory on tape. Once more, the number of units of the address are entered doesn't matter, but if more than 4 units, the lowest ranking 4 units are significant. When data is being entered on the tape, the cursor will turn on and off at the far left side and when finished, an asterisk (*) will appear-ready for a command. During W command execution, the remote ON will activate (Refer to TAPE use).

(4) L(Load Tape)

This command is used when loading the memory with data recorded on tape.

***L** **Return** after Key, enter L. When loading data from the tape, the cursor will blink on and off on the far left side, and when finished an asterisk will appear readying for a command. During L command execution,

remote output turns on. In the case of a tape read error, a question mark (?) will appear, and an entry command is awaited.

(5) LV (Load Verify Tape)

This command is used for comparing memory contents with the contents written on the tape.

*LV **Return** after Key, enter L&V. When loading data from tape, the cursor will blink off and on on the left side and when finished, an asterisk (*) will appear and another command is awaited. During LV command execution the remote output comes on.

Should a tape read "error" or a discrepancy between the two occur, a question mark (?) will be shown.

(6) G(Jump)

This command used when the operator wishes to jump to a certain address.

*Gxxxx **Return** (xxxx=jump address)

After Key, enter G and the jump address. Still once more, it matters not how many units are entered, but if over 4 units, only the lowest ranking address units will be significant.

(7) ^B (return to BASIC)

This command is used when the operator wishes to return from the monitor to BASIC.

*^B **Return** Hold down the **CRTL** key and enter the B key. When returned to BASIC, an "OK" will appear.

(8) TM (Test Memory)

This command is used to test the memory.

*TM **Return** After Key, enter TM. During this command's execution, the cursor will blink on and off at the far left side, and when the process is finished (and there are no errors), the system returns to BASIC and

responds with an "OK". (The characters and letters displayed on the surface are random.

Note: When the TM command is executed, program that was in memory can change. If there is a memory error, the bell will ring and the process will stop. If there is an error after the reset switch is pressed, the monitor is moved via the MON command, and the operator looks at the contents of the following addresses:

Address	Contents
FF39-FF3A	memory address (FF39-low rank address) in error (FF3A-high rank address)
FF3B numbers	data written in those addresses
FF3C numbers	data loaded addresses

So when the contents of the FF3B numbers are compared with the FF3C numbers, the bad bit will be identified.

Note: As for the contents of the FF39 through FF3C numbers, since only under normal circumstances does TL memory of the FF39-FF3C numbers become significant, after the TM command is entered in a situation in which an error occurs for a few seconds. These contents are still reliable.

Execution time
approximately 3 min. 30 sec.

SECTION IV

APPENDICES

APPENDIX A

Disk Files

A.0 Files

Data for disk operations is handled in units of "files". Files are made up of records, which are 256 bytes in length (one record corresponds to one sector). I/O is performed in units of "clusters", which consist of 8 records, allowing high speed data transfer.

Files are created under a user given name, and future references to the file are made under that name. It is not normally necessary to be concerned with a file's actual physical location.

BASIC disk files can be divided into program files, which are created by the program, and data files, which are used as data when the program is executed. Disk files can also be divided into sequential files and random files, according to the method by which they are accessed.

Maximum file size is 544 records. A file containing 544 records would be 139K bytes long, which is the total storage available on one disk. Up to 68 files may be stored on one disk, so long as their total length does not exceed 139K bytes.

A.1 Filenames

The format for filenames is:

`<filename>[,<file type>]`

`<filename>` consists of six letters, and if included, `<file type>` is separated by a period and consists of three letters. If less than six letters are specified as `<filename>`, spaces will be added to make the name up to six characters. `<filename>` is the name under which the file is referenced, and `<file type>` is used to categorize files. For example, to

indicate that a file is a BASIC program file, it can be given a name of the form: <filename>.BAS.

If more than six characters are specified as a filename, a period will be placed between the sixth and seventh character, the seventh to ninth character will automatically be interpreted as the file type, and characters after the ninth character will be dropped.

A.2 Program Files

Program files are formed under a name specified in a SAVE command. The LOAD, MERGE, and RUN statements affect only program files. KILL, NAME, and (L)FILES are applicable to both program files and data files. For details see the respective sections of each of these commands in Chapter 2.

A.3 Data Files

Data files formed under N-BASIC consist of sequential files and random files. The basic difference in these files is not so much their structure, but rather the manner in which they are accessed.

A.3.1 Sequential Files

Sequential files are easier to create than random files, but are limited in flexibility and speed when it comes to accessing the data. The data that is written to a sequential file is stored one item after another in the order it is sent, and is read back in the same way; whereas, data may be written or read from a random file starting from any location.

The statements and functions that are used with sequential files are: OPEN, PRINT #, INPUT #, PRINT # USING, LINE INPUT #, CLOSE, EOF, and LOC.

Sequential files are formed according to the following steps:

Operation	Example
1. Open file	OPEN "DATA" AS #1
2. Write data to file using the PRINT # or PRINT # USING statement.	PRINT #1,A\$,B\$,C\$
3. Close and re-open file. This step is necessary to read data from a sequential file.	CLOSE #1 OPEN "DATA" AS #1
4. Read data from file to program by execution of the INPUT # or LINE INPUT # statement.	INPUT #1,S\$,Y\$,Z\$

```

10 OPEN "DATA" AS #1
15 FOR I = 1 TO 4
20 INPUT "NAME";N$
30 INPUT "SECTION";D$
40 INPUT "DATA";H$
50 PRINT #1,N$;",";D$;",";H$
60 PRINT
70 NEXT I
(Program A-1)

```

```

RUN
NAME?          MICKEY MOUSE
SECTION?       ADVERTISEMENT
DATE?          79/04/01

NAME?          SHERLOCK HOMES
SECTION?       INVESTIGATION
DATE?          47/09/15

```

NAME? AUDREY HEPBURN
SECTION? PUBLIC RELATIONS
DATE? 52/01/14

NAME? SUPERMAN
SECTION? SECURITY
DATE? 79/12/31

OK

The next program will read the data file formed by program A-1 and print those items with a date of '79.

```
10 OPEN "DATA" AS #1
20 INPUT #1, N$, D$, H$
30 IF LEFT$ (H$,2) = "79" THEN PRINT N$
40 GOTO 20
```

RUN
MICKEY MOUSE
SUPERMAN
Input past end in 20

(Program A-2)

Program A-2 reads all items in the file sequentially. After all items have been read, the next execution of line 20 will cause an "Input past end" error. To avoid this, an EOF statement should be used.

```
10 OPEN "DATA" AS #1
15 IF EOF (1) THEN END
20 INPUT #1, N$, D$, H$
30 IF LEFT$ (H$,2) = "79" THEN PRINT N$
40 GOTO 15
```

By using the **PRINT#USING** statement, formatted files may be formed. For example, execution of the statement:

```
PRINT#1, USING "####.#",;A,B,C,D
```

will cause the commas, serving as delimiters, to be written to the disk along with the rest of the data. In this case, the final comma of the format statement is written to the disk as a data item delimiter.

When the **LOC** function is evaluated for a sequential file, the returned value is the number of sectors used since that file was opened. (One sector consists of 256 bytes).

A.3.1.1 Adding Data to Sequential Files

When adding data to a previously created sequential file, the file must be opened in the **"APPEND"** mode. For example, if line 10 of program A-1 is replaced with:

```
10 OPEN "DATA" FOR APPEND AS#1
```

the program will add data to the previously created file name **"DATA"**. Next the appended data can be checked by executing program A-2. When a file is opened in the **APPEND** mode, the file's mode is set to **APPEND**, and the record number is set to the record number of the last record of the file being appended to. Following statements in the program may move the pointer anywhere within the file. When the last record of the file is read, the file's mode is reset to **FILE**, and its pointer is left pointing at its last record. If it is necessary to add data to a file in this state, execute the statement:

```
GET#n, LOF(n)
```

This statement sets the file's pointer to the end of the file as preparation for appending.

A.3.2 Random Files

Creating and accessing random files requires more program steps than sequential files, but there are advantages to using random files. One advantage is that random files require less room on the disk, because BASIC stores them in a packed binary format. (A sequential file is stored as a series of ASCII characters).

The biggest advantage to random files is that data can be accessed randomly, i.e., anywhere on the disk — it is not necessary to read through all the information, as with sequential files. This is possible because the data is stored and accessed in distinct units called records and each record is numbered.

The statements and functions that are used with random files are:

OPEN	FIELD	LSET/RSET	GET
PUT	CLOSE	LOC	LOF
MKI\$	CVI		
MKS\$	CVS		
MKD\$	CVD		

A.3.2.1 Creating Random Files

The following steps are required for creating random files:

Operation	Example
1. Open File	OPEN "FILE" AS #1
2. Execute a FIELD statement to allocate random variable space to string variables.	FILE #1, 20 AS N\$, 4 AS A\$, 8 AS P\$

3. Execute a LSET statement to move the data to the random buffer. Before transferring numeric values to a random buffer, they must be converted to strings. Integers, single precision numbers, and double precision numbers may be converted to strings with MKI\$, MKS\$, and MKD\$ statement respectively.
4. Execute a PUT statement to transfer buffer contents to the disk.
- LSET N\$ = X\$
LSET A\$ = MKS\$(A)
LSET P\$ = TEL\$

PUT #1, CODE%

Program A-4 writes data entered from the keyboard to a random file. In this program, data is written to the file when the PUT statement in line 100 is executed. The two digit value input in line 30 becomes the record number.

```
10 OPEN "FILE" AS #1
20 FIELD #1, 20 AS N$, 4 AS A$, 8 AS A$, 8 AS P$
30 INPUT "2 digit code"; CODE%
40 INPUT "name"; X$
50 INPUT "amount"; AMT
60 INPUT "telephone"; TEL$: PRINT
70 LSET N$ = X$
80 LSET A$ = MKS$ (AMT)
90 LSET P$ = TEL$
100 PUT #1, CODE%
110 GOTO 30
```

(Program A-4)

A.3.2.2 Accessing Random Files

The following steps are necessary for accessing random files:

Operation	Example
1. Open File	OPEN "FILE" AS #1
2. Execute a FIELD statement to allocate random buffer space to string variables. Note: In programs which perform both input and output on the same random buffer, only one OPEN statement and one FIELD statement may be used.	FIELD #1, 20 AS N\$, 4 AS A\$, 8 AS P\$
3. Read the desired record into the random buffer with a GET statement.	GET #1, CODE%
4. The program can now access the data within the random buffer. Integer, single precision numeric, and double precision numeric data items can be changed from their string back to their numeric form by the CVI, CVS, and CVD functions, respectively.	PRINT N\$ PRINT CVS(A\$)

Program A-5 reads data from the random file "FILE" created by program A-4. Program A-5 accepts 2-digit codes, locates the corresponding data item in file "FILE", and prints the item.

```

10 OPEN "FILE" AS #1
20 FIELD #1, 20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2-digit code"; CODE%
40 GET #1, CODE%
50 PRINT N$
60 PRINT USING "$$ ###. ##"; CVS(A$)
70 PRINT P$: PRINT
80 GOTO 30

```

(Program A-5)

When the LOC function is evaluated for a random file, the present record number is returned. This is useful in checking the value of the record number when several GET or PUT statements with the record numbers omitted have been executed.

A.4 Initializing New Disks

The following steps must be performed before using new disks which have not yet been formatted or disks which are formatted for other systems.

1. Execute the **FORMAT** command to perform level 1 formatting. After performing this formatting, the PC-8031 can perform I/O, but **BASIC** files cannot be formed.
2. Perform level 2 formatting by means of a **BASIC** formatting program.

After these steps have been completed, **BASIC** files may be formed on a disk; however, improper operation, such as opening the drive's door while in operation, will ruin the formatting, and these steps must be carried out again.

After placing a disk for **BASIC** files into a drive, always execute the **MOUNT** command. Also, before removing a disk containing **BASIC** files from a drive, always execute the **REMOVE** command. If a disk is removed without executing the **REMOVE** command, or disks are exchanged

without executing either the **REMOVE** or **MOUNT** command, the file allocation table will not properly be updated and data will be lost. Disks mistakenly handled in this way must be reformatted at level 2 before using again.

APPENDIX B

Machine Language Subroutines

The **USR** function allows **N-BASIC** programs to call user subroutines, written in machine language, in the same manner as it calls intrinsic functions.

3.1 Memory Allocation

Memory space for user subroutines written in machine language must be secured before the program is loaded. As shown in the memory map, the area which can be used is at the very top of user **RAM**. The size of the area is determined by the second parameter in the **CLEAR** statement. Since the top of user **RAM** is address 59903 (**E9FF**₁₆), subtract from this value the number of bytes required for machine language subroutines, and execute a **CLEAR** statement with this difference as the second parameter. This will cause **N-BASIC** to consider this value as the upper boundary of its program area, making all locations from that point up to **E9FF** available for use by machine language routines.

For example, to secure 100 bytes of memory for machine language subroutines, execute the command.

CLEAR 300, 59903-100

For details on the **CLEAR** command, see Section 2.3.

When machine language subroutines are called by **N-BASIC**, the stack pointer is set so as to allow the subroutine to use up to eight stack levels (16 bytes). If more stack area is required, the stack pointer should be saved, reset to an area allocated for machine language subroutines, and restored to its original value before returning to **N-BASIC**.

3.2 Calling the USR Function

The format of the USR function is:

USR <number> (argument)

<number> may be integer from 0 to 9, and (argument) may be any numeric or string expression.

A correspondence is formed between the USR function and a DEFUSR statement for which the same <number> parameter has been specified, and when the USR function is called, program execution jumps to the address specified in this DEFUSR statement.

When the USR function is called, a value is loaded into the A register to indicate the argument type. The values used and their meanings are:

Value in Register A	Argument Type
2	Two byte integer (2's complement)
3	String
4	Single precision floating point
8	Double precision floating point

If the argument type is numeric, the register pair [HL] contains the address of the floating point accumulator (FAC-3) where the actual value is stored.

When (argument) is an integer,

*FAC-3 contains the lower eight bits of the argument

*FAC-2 contains the upper eight bits of the argument

When (argument) is a single-precision floating point number,

*FAC-3 contains the lower eight bits of the significant value

*FAC-2 contains the middle eight bits of the significant value

*FAC-1 contains the upper seven bits of the significant value

*Bit 7 of FAC-1 is the sign bit (0 = +, 1 = -).

When (argument) is a double-precision floating point number,

*FAC-7 through FAC-4 contains the additional four bytes of the significant value.

(FAC-7 contains the lower eight bits).

When (argument) is a string, the DE register pair points to the three-byte "string descriptor". Byte zero of the string descriptor contains the length of the string (0 to 255 bytes), and bytes one and two contain the address of the first byte of the string.

Note: When (argument) is a string contained within the program, the DE register pair points at the string. In such a case, there is the possibility that a string within a BASIC program will be incorrectly handled or destroyed by the subroutine, causing unpredictable execution results. To avoid this, the character combination: + " " may be added to a string. For example:

A\$ = "N-BASIC" + " "

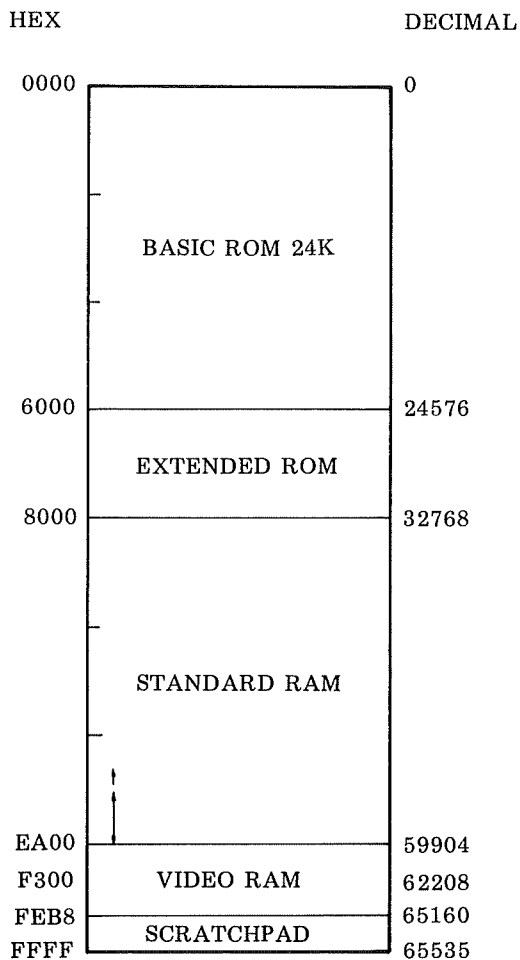
Since this string will now be copied to the string space in memory, its copy can be referenced.

Normally, the type of value returned from a user subroutine is the same as the argument specified for the user subroutine. However, an integer value may always be returned in the HL register pair by calling the MAKINT routine. The procedure for calling this routine is as follows:

PUSH H	;SAVE RETURN VALUE
LHLD 24H	;HEAD ADDRESS OF MAKINT
XTHL	;SAVE IN STACK, RESTORE
	;CONTENTS OF HL
RET	;RETURN

APPENDIX C

PC-8001 Memory Map



1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100.

APPENDIX D

Error Messages

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|------------------------------|---|
| 1 | NEXT without FOR | FOR and NEXT statements do not correspond properly. (Too many NEXT statements) |
| 2 | Syntax error | A line has been encountered that contains an incorrect sequence of characters (such as a misspelled statement). |
| 3 | RETURN without GOSUB | A RETURN statement has been encountered for which there is no previous, unmatched GOSUB statement. |
| 4 | Out of Data | A READ statement has been executed when there are no DATA statements with unread data remaining in the program. |
| 5 | Illegal function call | A parameter that is out of range has been passed to a match or string function. |

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|------------------------|---|
| 6 | Overflow | The result of calculation is too large to be represented in N-BASIC. |
| 7 | Out of memory | A program is too large or has too large array variables. |
| 8 | Undefined line number | A reference has been made to a line number that does not exist. |
| 9 | Subscript out of range | An array element has been referenced with a subscript that is outside the dimension of the array. |
| 10 | Redimensioned array | Two DIM statements have been given for the same array. |
| 11 | Division by zero | Division by zero has been encountered in an expression. |
| 12 | Illegal direct | A statement that is illegal in direct mode has been entered as a direct mode command. |
| 13 | Type mismatch | A derived value type does not match that assigned. |

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|----------------------------|--|
| 14 | Out of string space | String variables exceed the amount of allocated string space. |
| 15 | String too long | A string is too long (more than 255 characters). |
| 16 | String formula too complex | A string expression is too long or too complex. (An illegal nesting level of parentheses, etc.). |
| 17 | Can't continue | A non-executable CONT command has been encountered. (The pointer is destroyed, etc.). |
| 18 | Undefined User Function | A user function has been called before the function definition (DEF statement) is given. |
| 19 | No RESUME | An error trapping routine with no RESUME statement has been encountered. |
| 20 | RESUME without error | A RESUME statement has been encountered before an error trapping routine is entered. |

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|-------------------------------|--|
| 21 | Unprintable error | An error message is not available for the error condition which exists. |
| 22 | Missing Operand | An expression contains an operator with no operand following it. |
| 23 | Line buffer overflow | An attempt has been made to input a line that has too many characters. |
| 24 | Position not on Screen | Specified cursor location is out of screen range. |
| 25 | Bad File Data | File data on disk is improperly formatted. |
| 26 | Disk BASIC Feature | An attempt has been made to execute a disk command with no disk connected. |
| 27 | Communication Buffer Overflow | I/O buffer for peripheral devices has overflowed. |
| 28 | Port not initialized | The LSI in the port interface is not initialized. |
| 29 | Tape read ERROR | An error has been found in input from a cassette tape. |

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|----------------------|----------------|
|-----------------------|----------------------|----------------|

Error Message for Disk Functions

(If a disk units is not attached, error messages with codes 21 and 26 will be printed.)

| | | |
|-----------|--------------------------|---|
| 50 | Field overflow | A FIELD statement is attempting to allocate more than 255 characters of string variables. |
| 51 | Internal error | An internal malfunction has occurred in N-BASIC for disk. |
| 52 | Bad file number | A statement or command has referenced a file with a file number that is not OPEN . |
| 53 | File not found | A LOAD , KILL or OPEN statement has referenced a file that does not exist on the current file. |
| 54 | Bad file mode | An attempt has been made to access a sequential file as random file or vice versa. |
| 55 | File already open | An OPEN or KILL statement has been issued for a file that is already OPENed . |

| <u>ERROR
CODE</u> | <u>ERROR MESSAGE</u> | <u>MEANING</u> |
|-----------------------|--------------------------|---|
| 56 | Disk not mounted | An attempt has been made to access a disk which has not been MOUNTed. |
| 57 | Disk I/O error | A disk I/O error which cannot be recovered from has occurred. |
| 58 | File already exists | The filename specified in a NAME statement is identical to a filename already in use on the disk. |
| 59 | Disk already mounted | A MOUNT command was executed on a drive that is already MOUNTed. |
| 60 | Disk Full | All disk storage space is in use. |
| 61 | Input past end | An INPUT statement has been executed after all the data in the file has been INPUT. |
| 62 | Bad filename | An illegal form has been used for the filename. |
| 63 | Direct statement in file | A direct statement has been encountered while LOADing an ASCII-format file. |

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|----------------------|---|
| 64 | Bad allocation table | Allocation table has been damaged. |
| 65 | Bad drive number | A drive was specified that is not in the system. |
| 66 | Bad track/sector | A track/sector number is out of range. |
| 67 | Deleted record | Attempt was made to read a deleted record. |
| 68 | Rename across disks | A NAME command specified 2 different drives. |
| 69 | Sequential after PUT | A sequential write as attempted after a PUT to a random file. |
| 70 | Sequential I/O only | A sequential file was OPENed as a random file. |
| 71 | File not OPEN | An attempt was made to access a file that had not previously been OPENed. |
| 72 | File write protected | An attempt to write a protected file was made. |

| ERROR
CODE | ERROR MESSAGE | MEANING |
|-----------------------|----------------------|--|
| 73 | Disk offline | Disk is removed from system. Check the disk interface. |

APPENDIX E

ASCII Character Set

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

| | |
|---|--|
| 0 | |
|---|--|

| | |
|--|--|
| | |
|--|--|

| | |
|---|--|
| 1 | |
|---|--|

| | |
|--|----------------|
| | S _H |
|--|----------------|

| | |
|---|--|
| 2 | |
|---|--|

| | |
|--|----------------|
| | S _X |
|--|----------------|

| | |
|---|--|
| 3 | |
|---|--|

| | |
|--|----------------|
| | E _X |
|--|----------------|

| | |
|---|--|
| 4 | |
|---|--|

| | |
|--|----------------|
| | E _T |
|--|----------------|

| | |
|---|--|
| 5 | |
|---|--|

| | |
|--|----------------|
| | E _Q |
|--|----------------|

| | |
|---|--|
| 6 | |
|---|--|

| | |
|--|----------------|
| | A _K |
|--|----------------|

| | |
|---|--|
| 7 | |
|---|--|

| | |
|--|----------------|
| | B _L |
|--|----------------|

| | |
|---|--|
| 8 | |
|---|--|

| | |
|--|----------------|
| | B _S |
|--|----------------|

| | |
|---|--|
| 9 | |
|---|--|

| | |
|--|----------------|
| | H _T |
|--|----------------|

| | |
|----|--|
| 10 | |
|----|--|

| | |
|--|----------------|
| | L _F |
|--|----------------|

| | |
|----|--|
| 11 | |
|----|--|

| | |
|--|----------------|
| | H _M |
|--|----------------|

| | |
|----|--|
| 12 | |
|----|--|

| | |
|--|----------------|
| | C _L |
|--|----------------|

| Decimal Code | Character |
|--------------|--------------------------|
| 13 | <div>CR</div> |
| 14 | <div>SO</div> |
| 15 | <div>SI</div> |
| 16 | <div>DE</div> |
| 17 | <div>D₁</div> |
| 18 | <div>D₂</div> |
| 19 | <div>D₃</div> |
| 20 | <div>D₄</div> |
| 21 | <div>NK</div> |
| 22 | <div>SN</div> |
| 23 | <div>EB</div> |
| 24 | <div>CN</div> |
| 25 | <div>EM</div> |
| 26 | <div>SB</div> |
| 27 | <div>EC</div> |
| 28 | <div>→</div> |

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

29



30



31



32



33



34



35



36



37



38



39



40



41



42



















43



44



| Decimal Code | Character |
|--------------|---|
| 45 |  |
| 46 |  |
| 47 |  |
| 48 |  |
| 49 |  |
| 50 |  |
| 51 |  |
| 52 |  |
| 53 |  |
| 54 |  |
| 55 |  |
| 56 |  |
| 57 |  |
| 58 |  |
| 59 |  |
| 60 |  |

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

61

=

62

>

63

?

64

@

65

A

66

B

67

C

68

D

69

E

70

F

71

G

72

H

73

I

74

J

75

K

76

L

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

77

M

78

N

79

O

80

P

81

Q

82

R

83

S

84

T

85

U

86

V

87

W

88

X

89

Y

90

Z

91

[

92

/

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

93



94



95



96



97



98



99



100



101



102



103



104



105



106



107



108



| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

109

m

110

n

111

o

112

p

113

q

114

r

115

s

116

t

117

u

118

v

119

w

120

x

121

y

122

z

123

{

124

'

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

| | |
|-----|---|
| 125 |  |
|-----|---|

| | |
|-----|---|
| 126 |  |
|-----|---|

| | |
|-----|---|
| 127 |  |
|-----|---|

The following are non-standard codes used in the PC-8001.

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

| | |
|-----|---|
| 128 |  |
|-----|---|

| | |
|-----|---|
| 129 |  |
|-----|---|

| | |
|-----|---|
| 130 |  |
|-----|---|


| | |
|-----|---|
| 131 |  |
|-----|---|

| | |
|-----|--|
| 132 |  |
|-----|--|

















| | |
|-----|---|
| 133 |  |
|-----|---|

| | |
|-----|---|
| 134 |  |
|-----|---|

| | |
|-----|---|
| 135 |  |
|-----|---|

| | |
|-----|---|
| 136 |  |
|-----|---|

| | |
|-----|---|
| 137 |  |
|-----|---|

| Decimal Code | Character |
|--------------|---|
| 138 |  |
| 139 |  |
| 140 |  |
| 141 |  |
| 142 |  |
| 143 |  |
| 144 |  |
| 145 |  |
| 146 |  |
| 147 |  |
| 148 |  |
| 149 |  |
| 150 |  |
| 151 |  |
| 152 |  |
| 153 |  |

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

154



155



156



157



158



159



160



161



162



163



164



165



166



167



















168



169



| Decimal Code | Character |
|--------------|---|
| 170 |  |
| 171 |  |
| 172 |  |
| 173 |  |
| 174 |  |
| 175 |  |
| 176 |  |
| 177 |  |
| 178 |  |
| 179 |  |
| 180 |  |
| 181 |  |
| 182 |  |
| 183 |  |
| 184 |  |
| 185 |  |

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

| | |
|-----|----------|
| 186 | Ω |
|-----|----------|

| | |
|-----|----------|
| 187 | Γ |
|-----|----------|

| | |
|-----|-----|
| 188 | 0 |
|-----|-----|

| | |
|-----|----------|
| 189 | δ |
|-----|----------|

| | |
|-----|----------|
| 190 | κ |
|-----|----------|

| | |
|-----|----------|
| 191 | Σ |
|-----|----------|

| | |
|-----|----------|
| 192 | α |
|-----|----------|

| | |
|-----|-------|
| 193 | ν |
|-----|-------|

| | |
|-----|----------|
| 194 | Δ |
|-----|----------|

| | |
|-----|---------|
| 195 | β |
|-----|---------|

| | |
|-----|-------|
| 196 | ξ |
|-----|-------|

| | |
|-----|--------|
| 197 | η |
|-----|--------|

| | |
|-----|----------|
| 198 | θ |
|-----|----------|

| | |
|-----|-----|
| 199 | 1 |
|-----|-----|

| | |
|-----|-------|
| 200 | \pm |
|-----|-------|

| | |
|-----|-------|
| 201 | ν |
|-----|-------|

| Decimal Code | Character |
|--------------|--------------|
| 202 | π |
| 203 | \wedge |
| 204 | 2 |
| 205 | \oplus |
| 206 | $-$ |
| 207 | τ |
| 208 | ϕ |
| 209 | ω |
| 210 | \approx |
| 211 | \checkmark |
| 212 | 7 |
| 213 | 8 |
| 214 | 9 |
| 215 | ϵ |
| 216 | ϕ |
| 217 | \nexists |

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

218



219



220



221



222



223



224



225



226



227



228



229



230



231



















232



233



| Decimal Code | Character |
|--------------|---|
| 234 |  |
| 235 |  |
| 236 |  |
| 237 |  |
| 238 |  |
| 239 |  |
| 240 |  |
| 241 |  |
| 242 |  |
| 243 |  |
| 244 |  |
| 245 |  |
| 246 |  |
| 247 |  |
| 248 |  |
| 249 |  |

| Decimal Code | Character |
|--------------|-----------|
|--------------|-----------|

250

☐

251

☐

252

☐

253

☐

254

☐

255

☐

Character Code Chart

upper 4 bits

lower 4 bits

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|----|----|----|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | DE | | 0 | @ | P | | p | | | | ∞ | α | ¢ | | ⊗ |
| 1 | SH | D1 | ! | 1 | A | Q | a | q | | | ≥ | 3 | ν | ω | | |
| 2 | SX | D2 | " | 2 | B | R | b | r | | | · | γ | Δ | ≈ | | |
| 3 | EX | D3 | # | 3 | C | S | c | s | | | * | 4 | β | ✓ | | |
| 4 | ET | D4 | \$ | 4 | D | T | d | t | | | ≤ | 5 | ξ | 7 | | |
| 5 | EQ | NK | % | 5 | E | U | e | u | | | / | 6 | η | 8 | | |
| 6 | AK | SN | & | 6 | F | V | f | v | | | · | ε | θ | 9 | | |
| 7 | BL | EB | ▼ | 7 | G | W | g | w | | | ↑ | ρ | 1 | ι | | |
| 8 | BS | CN | (| 8 | H | X | h | x | | | ½ | σ | ± | φ | ♠ | |
| 9 | HT | EM |) | 9 | I | Y | i | y | | | ↓ | ψ | υ | ≠ | ♥ | |
| A | LF | SB | * | : | J | Z | j | z | | | ← | Ω | π | χ | ♦ | |
| B | HM | EC | + | ; | K | [| k | { | | | → | Γ | ∧ | ° | ♣ | |
| C | CL | → | , | < | L | \ | l | : | | | + | 0 | 2 | ⊗ | ● | |
| D | CR | ← | — | = | M |] | m | } | | | (| δ | ⊕ | ζ | ○ | |
| E | SO | ↑ | . | > | N | ^ | n | ~ | | |) | κ | — | λ | ◁ | |
| F | SI | ↓ | / | ? | O | _ | o | | | | ¼ | Σ | τ | μ | ▷ | |

APPENDIX F

Derived Functions

Functions that are not intrinsic to N-BASIC may be calculated as follows: (Consideration should be given as to the degree of accuracy.)

| FUNCTION | N-BASIC EQUIVALENT |
|--------------------|---|
| SECANT | $\text{SEC}(X)=1/\text{COS}(X)$ |
| COSECANT | $\text{CSC}(X)=1/\text{SIN}(X)$ |
| COTANGENT | $\text{COT}(X)=1/\text{TAN}(X)$ |
| INVERSE SINE | $\text{ARCSIN}(X)=\text{ATN}(X/\text{SQR}(-X*X+1))$ |
| INVERSE COSINE | $\text{ARCCOS}(X)=\text{ATN}(X/\text{SQR}(-X*X+1))+1.5708$ |
| INVERSE SECANT | $\text{ARCSEC}(X)=\text{ATN}(X/\text{SQR}(X*X-1))+(\text{SGN}(X)-1)*1.5708$ |
| INVERSE COSECANT | $\text{ARCCSC}(X)=\text{ATN}(X/\text{SQR}(X*X-1))+(\text{SGN}(X)-1)*1.5708$ |
| INVERSE COTANGENT | $\text{ARCCOT}(X)=\text{ATN}(X)+1.5708$ |
| HYPERBOLIC SINE | $\text{SINH}(X)=(\text{EXP}(X)-\text{EXP}(-X))/2$ |
| HYPERBOLIC COSINE | $\text{COSH}(X)=(\text{EXP}(X)+\text{EXP}(-X))/2$ |
| HYPERBOLIC TANGENT | $\text{TANH}(X)=\text{EXP}(-X)/(\text{EXP}(X)+\text{EXP}(-X))*2+1$ |
| HYPERBOLIC SECANT | $\text{SECH}(X)=2/(\text{EXP}(X)+\text{EXP}(-X))$ |

| FUNCTION | N-BASIC EQUIVALENT |
|------------------------------|--|
| HYPERBOLIC COSECANT | $\text{CSCH}(X)=2/(\text{EXP}(X)-\text{EXP}(-X))$ |
| HYPERBOLIC COTANGENT | $\text{COTH}(X)=\text{EXP}(-X)/(\text{EXP}(X)-\text{EXP}(-X))*2+1$ |
| INVERSE HYPERBOLIC SINE | $\text{ARCSINH}(X)=\text{LOG}(X+\text{SQR}(X*X+1))$ |
| INVERSE HYPERBOLIC COSINE | $\text{ARCCOSH}(X)=\text{LOG}(X+\text{SQR}(X*X-1))$ |
| INVERSE HYPERBOLIC TANGENT | $\text{ARCTANH}(X)=\text{LOG}((1+X)/(1-X))/2$ |
| INVERSE HYPERBOLIC SECANT | $\text{ARCSECH}(X)=\text{LOG}((\text{SQR}(-X*X+1)+1)/X)$ |
| INVERSE HYPERBOLIC COSECANT | $\text{ARCCHSCH}(X)=(\text{LOG}((\text{SGN}(X)*\text{SGR}(X*X+1)+1)/X))$ |
| INVERSE HYPERBOLIC COTANGENT | $\text{ARCCOTH}(X)=\text{LOG}((X+1)/(X-1))/2$ |

APPENDIX G

Reserved Words

ABS
AND
ASC
ATN
ATTR\$
AUTO

BCD\$
BEEP

CDBL
CHR\$
CINT
CLEAR
CLOAD
CLOSE
CMD
COLOR
CONSOLE
CONT
COS
CSAVE
CSNG
CSRLIN
CVD
CVI
CVS

DATA
DATE\$
DEC
DEF
DEFDBL

DEFINT
DEFSNG
DEFSTR
DELETE
DIM
DSKF
DSKI\$
DSKO\$

ELSE
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP

FIELD
FILES
FIX
FN
FOR
FORMAT
FPOS
FRE

GET
GOSUB
GOTO

HEX\$

IF
IMP
INIT
INKEY\$

INP
INPUT
INSTR
INT
IRESET
ISET

KEY
KILL

LEFT\$
LEN
LET
LFILES
LINE
LIST
LISTEN
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET

MAT
MERGE
MID\$
MKD\$
MKI\$
MKS\$
MOD
MON
MOTOR
MOUNT

NAME
NEW
NEXT
NOT

OCT\$
ON
OPEN
OR
OUT

PEEK
POINT
POKE
POLL
PORT
POS
PRESET
PRINT
PSET
PUT

RBYTE
READ
REM
REMOVE
RENUM
RESTORE
RESUME
RETURN
RIGHT\$
RND
RSET
RUN

SAVE
SET
SGN

SIN
SPACE\$
SQR
SRQ
STATUS
STEP
STOP
STR\$
STRING\$
SWAP

TALK
TAN
TERM
THEN
TIME\$
TO
TROFF
TRON

USING
USR

VAL
VARPTR

WAIT
WBYTE
WIDTH

XOR

NOTES

NOTES

NOTES





NEC

Printed in Japan

78110341